

Programming Assignment 1

Getting Started with C++

Time due: 9:00 PM Tuesday, January 17th

The purpose of this assignment is to have you start learning how to use the Visual C++ and either Xcode or g++ environments and understand a variety of programming errors.

Here's what you are to do:

1. (optional) Obtain a copy of [Visual C++](#) and install it. You don't need to do this if you prefer to do your Visual C++ work on the SEASnet computers (in the lab or remotely). If you will be using another development environment (such as [Xcode on a Mac](#)), use this project to get comfortable using it as well as Visual C++.
2. Enter [this C++ program](#) into your development environment. Do not change the program.
3. Build the executable from the program. (Fix any typos you may have made when entering the program.)
4. Execute the program with a variety of reasonable input integers to see if it runs as one would expect from reading the source code. (If the Visual C++ console window disappears when your program finishes executing, before you have a chance to see the output, you probably forgot to do step 4 from the [Visual C++ writeup](#), or you started execution by selecting the Start Debugging item from the Debug menu or by double-clicking on the .exe file. What you want to do is select the Start Without Debugging item from the Debug menu; if there is no such menu item, fix it as directed in step 7 of the writeup.)
5. Using the program as given, run it with input integers that cause it to produce incorrect, unusual, or nonsensical results. (Notice we're saying to try input *integers*, not input like **124765.23** or **strawberry**.)
6. Starting from the program as given, introduce into the source code at least one error that someone might make that, while not preventing a successful build, causes the program when it runs to produce incorrect results from reasonable input.
7. Again starting from the program as given, introduce at least two distinct types of mistakes that someone might make, each of which would cause the program to fail to compile correctly.

You should create a separate project for each of steps 2, 6, and 7, since you're not allowed to have multiple files in the same project if more than one has a main routine.

In addition to running the programs under Visual C++, run them using clang++ or g++ (perhaps using [Xcode on a Mac](#) or [g++ with Linux](#)).

What you will turn in for this assignment is a compressed file in zip format containing exactly four files:

1. A file named **original.cpp** that contains the program as given.
2. A file named **logic_error.cpp** with the program you produced in step 6.
3. A file named **compile_error.cpp** with the program you produced in step 7.
4. A file named **report.doc** or **report.docx** (in Microsoft Word format) or **report.txt** (an ordinary text file) that describes the input you provided in step 5 and each of the errors you introduced into the logic_error.cpp and compile_error.cpp programs. Briefly discuss any error messages the compiler reported, and incorrect, unusual, or nonsensical results. This report may well end up being much less than a page long. Please be sure to include your **name** and your **UCLA Id number** in this document.

The zip file itself may be named whatever you like.

Do **not** include anything else in the zip file. (Some Windows users seem not to be aware of Windows filename extensions, so end up putting the wrong files in their zip file.) To create a zip file on a SEASnet machine, you can select the four files you want to turn in, right click, and select "Send To / Compressed (zipped) Folder". Under macOS, copy the files into a new folder, select the folder in Finder, and select File / Compress "*folderName*"; make sure you *copied* the files into the folder instead of creating aliases to the files.

We will be using software tools to help us grade your projects, so there are certain requirements you must meet for the tools to work: **The zip file you turn in for this project must have *exactly* four files in it, with *exactly* the names indicated. If you do not follow these requirements, your score on this project will be *zero*.** "Do you mean that if I do everything right except misspell a file name or include an extra file, I'll get no points whatsoever?" Yes. That seems harsh, but attention to detail is an important skill in this field. A draconian grading policy certainly encourages you to develop this skill. The only exception to the requirement that the zip file contain exactly four files of the indicated names is that if you create the zip file under macOS, it is acceptable if it contains the additional files that the macOS zip utility sometimes introduces: **MACOSX**, **.DS_Store**, and names starting with **._** that contain your file names. In an effort to assist CS 31 students with the contents of your .zip file archive,

Howard has created a [Zip File Checker](#) which will echo back to you the contents of your .zip file. Please use this file checker to ensure you have named all your files correctly.

By January 11, there will be links on the class webpage that will enable you to turn in your zip file electronically. Turn in the file by the due time above. Remember that most computing tasks take longer than expected; this applies especially to steps 1, 2, and 3 above. Start this assignment now!

Use this project as an opportunity to learn what happens when you make mistakes. After you've turned in what's required for this project, play around. Introduce a mistake into the program and see what happens. Fix it. Introduce a different mistake and see what happens then. Fix it. Keep doing this so you can see the kinds of problems that might arise when you develop your own programs and what the compilers say for each particular problem (if they even detect them at all). See what happens if you make more than one mistake in the program. Will they all be detected? Will an earlier mistake interfere with the reporting of a later one?

Here is the C++ program:

```
// Code for Project 1

// Electoral College

#include <iostream>

using namespace std; // pp. 37-38 in Savitch book explain this line

int main() {

    int totalVoters;

    int republicanVotes, democraticVotes;

    cout << "How many Electoral College electors are there? ";

    cin >> totalVoters;

    cout << "How many of these electors voted for the Republican party candidate? ";

    cin >> republicanVotes;

    cout << "How many of these electors voted for the Democratic party candidate? ";

    cin >> democraticVotes;

    double pctRepub = ( 100.0 * republicanVotes ) / totalVoters;

    double pctDemoc = ( 100.0 * democraticVotes ) / totalVoters;
```

```
cout.setf(ios::fixed); // see pp. 30-31 in Savitch book
cout.precision(1);

cout << endl;

cout << pctRepub << "% of the " << totalVoters << " electors voted for the Republican
party candidate." << endl;  cout << pctDemoc << "% of the " << totalVoters << " electors
voted for the Democratic party candidate." << endl;  cout << endl;

if (pctRepub > pctDemoc)
{
    cout << "Looks like the Republican party candidate won!" << endl;
}

if (pctDemoc > pctRepub)
{
    cout << "Looks like the Democratic party candidate won!" << endl;
}

if (republicanVotes + democraticVotes != totalVoters)
{
    cout << "Looks like some of these data values don't make sense." << endl;
}

return(0); }
```